DATE: 4/01/97

CONTROLLING OFFICE FOR THIS DOCUMENT IS:
DIRECTOR, Army High Performance Computing
Research Center (AHPCRC)
Army Research Laboratory
Aberdeen, MD

POC: Director (Tayn E. Tezduyar)

DISTRIBUTION STATEMENT A: Public release

Scalable Parallel Algorithms for Sparse Linear Systems

Wint 95

by Anshul Gupta (AHPCRC), George Karypis (AHPCRC) and Vipin Kumar (AHPCRC)

Large sparse linear systems occur in many scientific and engineering applications encountered in military and civilian domains. Such systems are typically solved using either iterative or direct methods. We are developing parallel formulations of computationally intensive algorithms that underly these methods.

Direct methods for solving sparse linear systems are important because of their generality and robustness. For linear systems arising in certain applications, such as linear programming and some structural engineering applications, they are the only feasible methods. Although highly parallel formulations of dense matrix factorization are well known, it has been a challenge to implement efficient sparse linear system solvers using direct methods, even on moderately parallel computers.

We have recently achieved a breakthrough in developing a highly parallel sparse Cholesky factorization algorithm that substantially improves the state of the art in parallel direct solution of sparse linear systems-both in terms of scalability and overall performance. Experiments have shown that this algorithm can easily speedup Cholesky factorization by a factor of at least a few hundred up to 1024 processors, and achieve levels of performance that were unheard of and unimaginable for this problem until very recently.

Matrix Name	Order	NonZeros	Description
BCSSTK30(B30)	28294	1007284	Off-shore generator platform
CSSTK31 (BC31)	35588	572914	Stiffness matrix of an automobile component
BCSSTK32(BC32)	44609	985046	Stiffness matrix of an automobile chassis
BRACK2 (BRCK)	62631	366559	Finite element mesh
CANT (CANT)	54195	1960797	Finite element mesh (3D)
COPTER2 (COPT)	55476	352238	Helicopter rotor mesh
CUBE35 (C35)	42875	124950	35 ´ 35 ´ 35 3D mesh
CYLINDER93(CY93)	45594	1786726	Finite element mesh (3D)
4ELT (4ELT)	15606	45878	NASA Airfoil (2D)
INPRO1 (INPR)	46949	1117809	Finite element mesh (3D)
MAROS-R7 (MR7)	3136	330472	Linear programming problem
NUG15 (NG15)	6330	186075	Quadratic assignment problem
ROTOR (ROTR)	99617	662431	Finite element mesh (3D)
SHELL93 (SHEL)	181200	2313765	Finite element mesh (3D)
TROLL (TROLL)	213453	5885829	Finite element mesh (3D)

WAVE (WAVE) 156317 1059331 Finite element mesh (3D)

Table 1. Description of test matrices used in our experiments.

It is a well known fact that dense matrix factorization scales well and can be implemented efficiently on parallel computers. We have shown that our parallel sparse factorization algorithm is asymptotically as scalable as the best dense matrix factorization algorithms on a variety of parallel architectures for a wide class of problems that include two and three-dimensional finite element problems. This algorithm incurs less communication overhead than any previously known parallel formulation of sparse matrix factorization, and therefore, is suitable for workstation clusters that tend to be connected via relatively low-bandwidth and high-latency channels relative to the traditional MPP platforms. We have successfully implemented this algorithm for Cholesky factorization on a variety of parallel computers, such as nCUBE2, CM-5, IBM SP-1 and SP-2, and the Cray T3D. The implementation on the T3D delivers up to 20 GFlops on 1024 processors for medium-size structural engineering and linear programming problems. Although our current implementations work for Cholesky factorization, the algorithm can be adapted for solving sparse linear least squares problems and for Gaussian elimination of diagonally dominant matrices that are almost symmetric in structure. Figure 1 shows the performance of our scheme on the matrices given in Table 1.

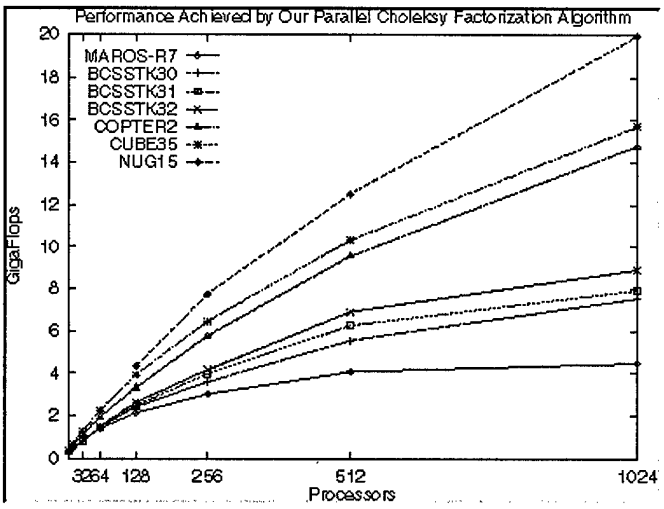

Figure 1. The performance of the parallel sparse multifrontal algorithm for various problems on the Cray T3D.

Fast and accurate graph partitioning algorithms are needed for the solution of sparse system of linear equations $Ax = b$ on a parallel computer. In the case of direct solvers, a graph partitioning algorithm can be used to reorder the matrix so that the amount of fill is minimized, and the concurrency that can be

exploited during parallel factorization is maximized. In the case of parallel iterative solvers, the graph corresponding to matrix A needs to be partitioned into p parts so that the number of edges with vertices on different partitions is minimized. Many heuristic algorithms are known for finding good partitions of a graph. Algorithms that provide good partitions of the graph (e.g., spectral methods) tend to be very slow, especially for large graphs. Faster algorithms tend to compromise on the quality of the partition. In the context of direct methods, good sequential partitioning methods can take even more time than the factorization step running on a parallel computer, and cheaper methods result in a high degree of fill in the matrix, causing overall factorization time to jump up by a large factor.

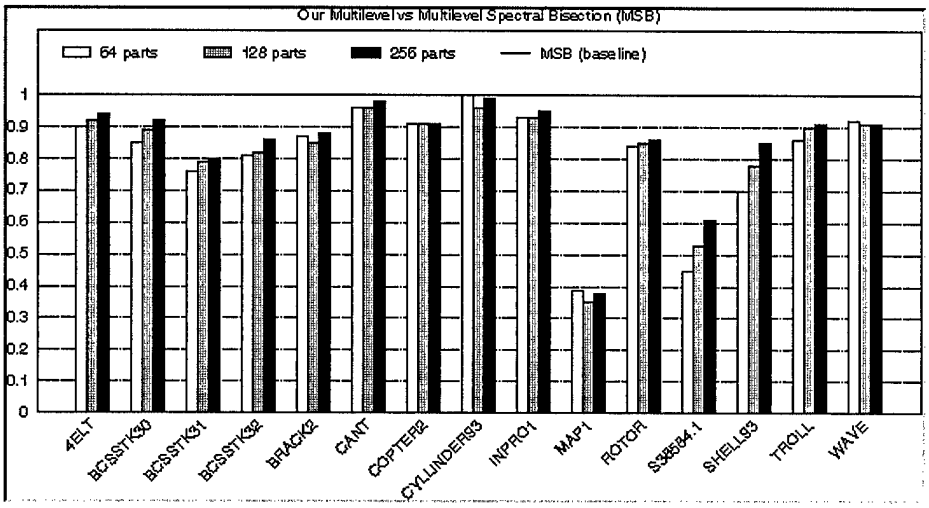

Figure 2. The size of the edge-cut of multilevel graph partitioning relative to spectral bisection. Bars below 1 indicate that the multilevel graph partitioning scheme produces better partitions.

We have recently developed a multilevel graph partitioning scheme that consistently outperforms the spectral partitioning schemes in terms of cut size and is substantially faster. We also used our graph partitioning scheme to compute fill reducing orderings for sparse matrices. Surprisingly, our scheme substantially outperforms the multiple minimum degree algorithm (MMD), which is the most commonly used method for computing fill reducing orderings of a sparse matrix. Figure 2 shows the performance of our multilevel scheme relative to the multilevel spectral bisection (MSB) on graphs corresponding to some of the matrices from Table 1. As the figure shows, the edge-cut produced by our multilevel scheme is consistently better than that produced by the MSB scheme.

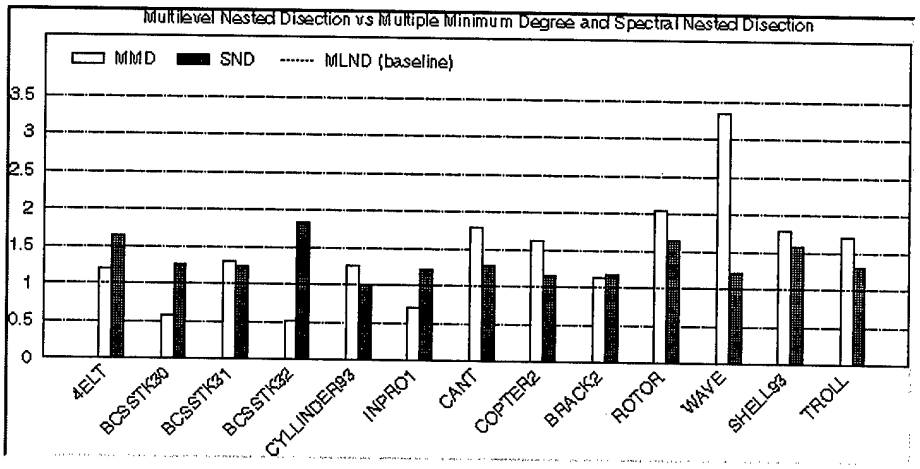

Figure 3. The number of operations required by spectral nested dissection and multiple minimum degree relative to multilevel nested dissection. Bars below 1 indicate that the multilevel scheme produces worse orderings.

Figure 3 shows the quality of the fill-reducing ordering produced by our multilevel scheme relative to the MMD scheme. From Figure 3 we see that our multilevel scheme does consistently better as the size of the matrices increases and as the matrices become more unstructured. When all test matrices are considered, MMD produces orderings that require a total of 702 billion operations, whereas the orderings produced by our multilevel scheme require only 293 billion operations. Thus, the entire ensemble of matrices can be factored roughly 2.4 times faster if ordered with our algorithm.

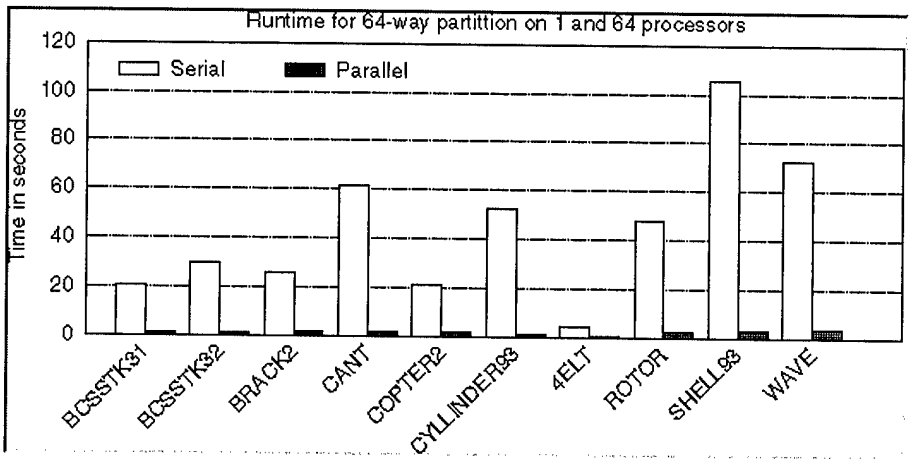

Figure 4. The speedup of graph partition by using the parallel algorithm.

Even though these multilevel algorithms are quite fast compared with spectral methods, performing a multilevel partitioning in parallel is desirable for the following reasons. With the recent development of highly parallel formulations of sparse Cholesky factorization algorithms, numeric factorization on parallel

computers can take much less time than the ordering step running on a serial computer. In the context of iterative methods, adaptive grid computations dynamically adjust the discretization of the physical domain. Such adjustments change the grid and thus require repartitioning of the graph. Being able to perform the partition in parallel is essential for reducing the overall run time of these types of applications. Furthermore, the amount of memory on serial computers is not large enough to allow the partitioning of graphs corresponding to large problems that can now be solved on massively parallel computers and workstation clusters. By performing graph partitioning in parallel, the algorithm can take advantage of the significantly higher amount of memory available in parallel computers.

We have recently developed a parallel formulation of the multilevel graph partitioning algorithm. Our parallel algorithm achieves a speedup of up to 56 on 128 processors for medium size problems, further reducing its already moderate serial run time. Figure 4 shows some of these results.